On a Parallel Implementation of Geometric Hashing on the Connection Machine [*]

Isidore Rigoutsos
Courant Institute of Mathematical Sciences

Robert Hummel
Courant Institute of Mathematical Sciences
& The Center for Neural Science

New York University
251 Mercer Street
New York, NY 10012
rigoutso@cs.nyu.edu hummel@cs.nyu.edu
Telephone: (212) 998 3471 FAX: (212) 995 4122

Abstract

We report on a scalable parallel implementation of geometric hashing on a Connection Machine. The algorithm that is employed has been described in [9]. Using the resulting implementation, it is possible to recognize models consisting of patterns of points embedded in scenes, independent of translation, rotation, and scale changes, when there are thousands of models containing approximately 16 points each, with scenes consisting of hundreds of points, where most of the scene points are spurious noise points, and where embedded model points in the scene may be obscured or misplaced. With 1024 models and a scene of 200 points, the implementation yields an execution time of 70 milliseconds per probe on a 64K-processor CM-2 parallel computer. Most of the execution time is taken in performing histograms of (model, basis-set) records. The algorithm is scalable, yielding an expected execution time that is $\mathcal{O}(\log^2 M + \log S \log M)$ on a Mn^3-processor hypercube-connected SIMD machine such as the Connection Machine. M is the number of models, n is the number of points per model, and S is the number of scene points. We also describe and report on a series of experiments for both the similarity and rigid transformation cases; these experiments provide information about detection and false alarm rates for varying amounts of noise in the input.

*This research was supported by AFAL contract F33615-89-1087, and by an IBM Graduate research Fellowship. Access to a Connection Machine was made possible through the DARPA Connection Machine Network Server Program. We thank Lew Tucker and Alan Mainwaring for discussions and encouragement. Haim Wolfson contributed to both the research and manuscript, and his assistance is gratefully acknowledged.

1 Introduction

In a model-based vision system, features such as edges, lines, line-endings, corners, and textures are extracted and localized in digital imagery, and compared with a database of models in order to identify, locate, and complete objects observed in a scene. Many model-based vision systems are based on hypothesizing matches between scene features and model features, predicting new matches, and verifying or changing the hypotheses through a search process [1, 3, 7]. A recent method, called *geometric hashing* [2, 4, 6], offers a different paradigm.

In geometric hashing, the collection of models are used in a preprocessing phase (executed "off-line" and only once) in order to build a *hash table* data structure. The data structure encodes the information about the models in a highly redundant multiple-viewpoint way. During the recognition phase, when presented with a scene and extracted features, the hash table data structure is used to index geometric properties of the scene features to candidate matching models. A search is still required over features in the scene. However, the geometric hashing scheme no longer requires a search over the features in the model sets. The result is that the recognition phase offers computational efficiencies over more traditional model-based vision methods. As we describe elsewhere [9], there is parallelism available in the search over scene features, and there is parallelism in the indexing process inherent within the hashing scheme. Further, the computational efficiencies afforded by geometric hashing translate into a reduction in the number of independent tasks that may be simultaneously conducted (at the expense of increased storage requirements), thereby decreasing the number of processors that are needed in a parallel implementation.

In this paper, we report on an implementation on the Connection Machine of one of the algorithms described in [9]. We also describe a number of modifications that are useful for efficient parallel implementation of the method for the particular case of point features. The last section of the paper presents the results of a number of experiments that were carried out using this implementation. These experiments had a two-fold purpose: first, we evaluated the real time performance of the implementation with a large database of models; second, the behavior of the method in the case of rigid or similarity transforms was examined, as a function of the noise present in the input. Other statistics are also presented.

Ultimately, we are able to achieve recognition of dot patterns embedded in scenes, independent of translation, rotation, and scale changes to the patterns, when there are thousands of models containing approximately 16 points each, with scenes consisting of hundreds of points, where most of the scene points are spurious noise points, and where embedded model points in the scene may be obscured or misplaced. The system will need to search over pairs of points in the scene (probes), and recognition will be obtained as soon as a pair of points is chosen so that both points lie on the embedded object, although multiple pairs may be used at the same time, and many heuristics exist for chosing likely basis pairs. We show that, using a randomized algorithm, no more than a few tens of probes are needed. With 1024 models and scenes consisting of 200 points, the implementation yields an execution time of approximately 70 milliseconds per probe, with a fixed basis set, on a 64K-processor CM-2 parallel computer. If multiple probes were used at once, the execution time would increase, but not in proportion to the number of basis pairs.

2 Foldings and Hash Table Equalization

In this section we discuss certain design issues for efficient parallel implementation of the geometric hashing algorithm. We discuss three issues:

- Modification of the coordinate calculation relative to a basis set;
- Requantization of the hash space for hash-bin frequency equalization; and
- Use of symmetries for reduction of the database size.

2.1 Coordinate Calculations

Once a basis set is selected, the location of the other points must be calculated relative to the basis set. As presented elsewhere [4, 6], a coordinate system is typically defined from a two-point basis set $\mathbf{e}_1$, $\mathbf{e}_2$ by placing the origin at $\mathbf{e}_1$, and orienting the system so that $\mathbf{e}_2$ lies on the positive x-axis. In our work, we define a coordinate system based on $\mathbf{e}_1$ and $\mathbf{e}_2$ so that the origin is located at the *midpoint* between $\mathbf{e}_1$ and $\mathbf{e}_2$ with $\mathbf{e}_2$ on the positive x-axis. Thus the (u, v) coordinates of $\mathbf{p}$ based on the basis set $\mathbf{e}_1$ and $\mathbf{e}_2$ are given by:

$$\mathbf{p} = u\,\mathbf{e}_x + v\,\mathbf{e}_y + \frac{\mathbf{e}_1 + \mathbf{e}_2}{2}$$

for the case of rigid transformations, where $\mathbf{e}_x = (\mathbf{e}_2 - \mathbf{e}_1)/\parallel \mathbf{e}_2 - \mathbf{e}_1 \parallel$ and $\mathbf{e}_y = \mathrm{Rot}_{90}(\mathbf{e}_x)$. For the case of similarity transformation, we use the same formula, except $\mathbf{e}_x = (\mathbf{e}_2 - \mathbf{e}_1)/2$ and $\mathbf{e}_y = \mathrm{Rot}_{90}(\mathbf{e}_x)$. Positioning the origin at the midpoint between $\mathbf{e}_1$ and $\mathbf{e}_2$ causes a shift in the u coordinate. The result is a more stable representation of the locations of the points, and, more importantly, permits a simpler exploitation of symmetries that occur due to combinations of the basis points.

2.2 Hash Space Quantization

A uniform distribution of the entries over the hash table is desirable in order to reduce execution time, and to efficiently store the hash table data structure. However, typical hash bin occupancies for uniformly quantized hash bins are nonuniform; Figure 1 shows a typical hash bin histogram.

We now describe a method to transform the coordinates of point locations so that the equispaced quantizer in hash space yields an expected uniform distribution. We must first determine the expected probability density $f(x, y)$ of the distribution of the untransformed coordinates. This can be done either by fitting a parametric model to available data or by calculating analytically the expected distribution based upon a model for the distribution of model points in the plane. We will assume that the model points are distributed either according to a uniform distribution over the unit disc or a Gaussian distribution over $\mathcal{R}^2$. We treat separately the cases where models can undergo a similarity transformation (rotation, translation, and scale), or simple rigid transformation (rotation and translation). From $f(x, y)$, we must calculate the probability density $f^*(u, v)$ of entries in hash space. Once the probability density $f^*(u, v)$ is known, then a transformation which maps that distribution to the uniform distribution over a rectangle can be computed. The mapping is effectively a *rehashing* function and can be used to evenly distribute the bin entries over the rectangular hash table.

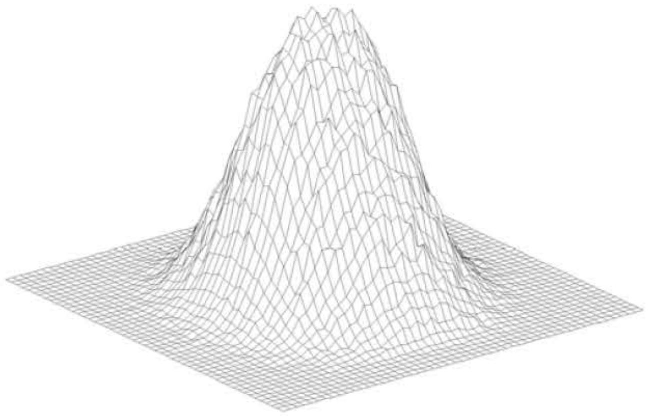


Figure 1. Hash bin occupancy for a typical database.

For the case where the model points (x_i, y_i) are distributed according to a Gaussian distribution with mean zero and standard deviation σ, the density functions in hash space can be computed analytically, yielding the following results for each of the transformation classes:

$$\text{For Rigid Transformations} \quad f^r(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{3\sigma^2}}, \tag{1}$$

$$\text{For Similarity Transformations} \quad f^s(u,v) = \frac{2}{\pi} \frac{1}{(u^2+v^2+3)^2}. \tag{2}$$

For the case of a uniform distribution of model points (x, y) over the unit disc, a solution proves to be too difficult to obtain analytically. However, we may apply the method of Levenberg-Marquardt [8] to synthetically generated data. We find that the distribution of transformed points in the case of rigid transformations can be approximated well by:

$$\text{For Rigid Transformations} \quad f^r(u,v) = \frac{1}{((4.7u^2+3.9v^2)^2+36.7)^2}. \tag{3}$$

A fit for the distribution of transformed points under similarity transformation has not been attempted.

For each of these three cases, a remapping function is easy to compute. the remapping function is a function $h(u, v)$, taking $\mathcal{R}^2$ to $\mathcal{R}^2$, that is applied to the transformed coordinates such that equally-spaced bins in the remapped space have uniform expected population. For the case of Equations 1, 2, and 3, the appropriate remapping functions are respectively:

$$h(u,v) = \left(1 - e^{-\frac{u^2+v^2}{3\sigma^2}}, \operatorname{atan2}(v,u)\right), \tag{4}$$

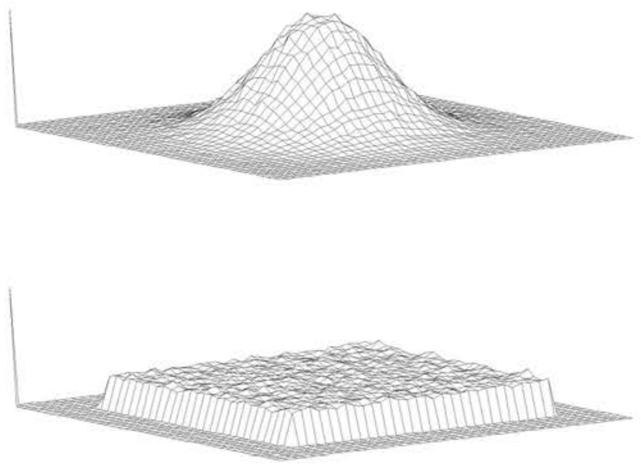


Figure 2. Hash table equalization for rigid transformations. Points are Gaussian distributed over $\mathcal{R}^2$.

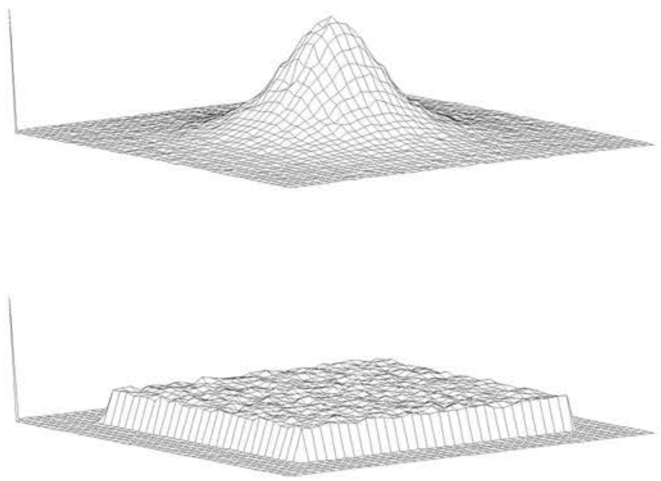


Figure 3. Hash table equalization for similarity transformations. Points are Gaussian distributed over $\mathcal{R}^2$.

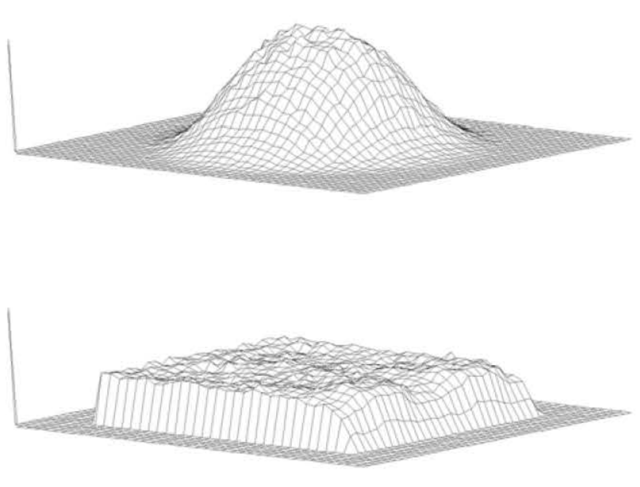


Figure 4. Hash table equalization for rigid transformations. Points are distributed over the unit disc.

$$h(u,v) = \left(1 - \frac{3}{u^2+v^2+3},\ \mathrm{atan2}(v,u)\right), \quad \textit{and} \qquad (5)$$

$$h(u,v) = \left(\frac{2}{\pi}\arctan\left(\frac{(au)^2+(bv)^2}{c^2}\right) + \frac{2[(au)^2+(bv)^2]\cdot c^2}{\pi([(au)^2+(bv)^2]^2+c^4)},\ \mathrm{atan2}(v,u)\right) \qquad (6)$$

respectively, and $a = 4.$[illegible], $b =$ [illegible], $c = ($[illegible]$6.$[illegible]$)^{1/}$[illegible].

Figures 2, 3 and 4 show the results of re-mapping each distribution $f^*(u,v)$ to a rectangular region using the remapping functions 4, 5 and 6. In all three cases, the data was created synthetically, by generating a collection of points $\{(x_i, y_i)\}$ according to the relevant distribution $f(x,y)$, and generating points (u,v) relative to basis sets (x_{i_1}, y_{i_1}), (x_{i_2}, y_{i_2}) by means of Monte Carlo sampling. As can be seen, the remappings are very efficient.

2.3 Symmetries and Foldings in the Hash Space

With the choice of the coordinate transformation as described in Section 2.1, and even if we use the remapping functions of Section 2.2, we discover certain symmetries in the storage pattern of entries in the hash table. The most prominent symmetry arises for the following reason.

If $\mathbf{e}_1$ and $\mathbf{e}_2$ form a basis pair of model m, and the coordinates of point $\mathbf{p}$ in model m are (u,v), then there will an entry of the form $(m, (\mathbf{e}_1, \mathbf{e}_2))$ at location (u,v) in the hash table (or at the rehashed position corresponding to (u,v)). When the basis pair $(\mathbf{e}_2, \mathbf{e}_1)$ is used, we discover that the coordinates of $\mathbf{p}$ are $(-u,-v)$. Thus in the location $(-u,-v)$ there will be an entry of the form $(m, (\mathbf{e}_2, \mathbf{e}_1))$. Due to the symmetry of the rehashing functions, the location of this entry, even when rehashing is used, will be symmetrically related to the entry of the form $(m, (\mathbf{e}_1, \mathbf{e}_2))$ caused by $\mathbf{p}$ when $(\mathbf{e}_1, \mathbf{e}_2)$ is used as a basis.

The result is that for every entry in the hash table (before remapping) above the $v = 0$ axis, there is an equivalent entry in the bottom half plane below the $v = 0$ axis, with the only change that the basis pair has been reversed. Thus, there is no need to store the bottom half of the hash table! Instead, whenever a hash occurs during the recognition phase to the lower half plane, the entries can be generated from the entries in the upper half plane.

A further possibility is to perform a *folding* of the hash space. For example, when a hash occurs to the lower half-plane, rather than generating the entries from the list of entries in the symmetrically opposite position of the upper half-plane, we can instead register a vote for the entire bin in the upper half-plane. This will confuse entries of the form $(m, (\mathbf{e}_1, \mathbf{e}_2))$ with entries of the form $(m, (\mathbf{e}_2, \mathbf{e}_1))$ — thus our basis pairs are now basis sets, and $(\mathbf{e}_1, \mathbf{e}_2)$ is the same as $(\mathbf{e}_2, \mathbf{e}_1)$. Although this means that a particular $(model, basis)$ may receive more votes than it deserves, we have encountered no difficulties with this method.

3 Data Parallel Algorithms

3.1 Data Parallel Implementation of Geometric Hashing

Since a principle strength of the geometric hashing algorithm is its parallelizability, a mapping of the algorithm to an available parallel architecture is important and enlightening. The level of available parallelism is fine-grain and thus our implementation is based on "data parallelism." Two data parallel algorithms for SIMD Hypercube architectures are described in detail in [9].

Data parallelism means that the separate tasks to be performed concurrently are indexed by items in the data structures that participate in the algorithm. In the case of geometric hashing, the items include the points in the image that compute hash locations, the hash bins, and the model/basis pairs that receive votes. Data parallelism is typically implemented on an SIMD machine (single instruction, multiple data); algorithm design and analysis typically uses the PRAM (parallel random access machine) model of computation. In our design of algorithms for implementing geometric hashing, we will at times consider ourselves limited to a hypercube-connected fine-grain SIMD parallel processor.

3.2 The Connection Machine

Our implementation makes use of a Connection Machine (model CM-2). The Connection Machine is a parallel SIMD machine supporting the data parallel model of computation. It consists of both hardware and software components. The hardware components of the system include a front end computer, a parallel processing unit with an array of up to 64K bit-serial processors, a high performance VME I/O bus, and a number of peripherals, such as framebuffers for displaying data and the DataVault$^{(TM)}$, for storage and fast retrieval of data structures.

3.3 Paris Interface

Paris is a low-level instruction set that allows users to write data parallel programs for the Connection Machine system using the front end as a host, regarding the parallel unit as a peripheral. The Paris facilities are available to the user through language interfaces such as C/Paris, Lisp/Paris, and Fortran/Paris. Each interface is essentially a subroutine library providing support for:

- *arithmetic operations* such as signed, unsigned, or floating point additions, subtractions, multiplications et cetera;
- *Boolean operations* such as logical AND, NOT, OR, and XOR;
- *data transfer* between processors and the front end; and
- *inter-processor communication*, including global sums and parallel prefix operations.

Other routines are provided to perform bookkeeping functions, such as maintaining pointers on the front end. We have used the C-Paris interface in order to implement the geometric hashing algorithm. Higher-level languages that are preprocessed to interface languages are available. However, we found it preferable to maintain fine control of the parallel operations through direct calls to the Paris facilities.

3.4 Virtual processors

A user on a CM-2 typically uses 8K, 16K, 32K or 64K processors simultaneously. For some applications, more processors would be desirable. In our case, when computing the histogram over model/basis pairs, we will want a separate processor for each possible pair, which amounts to at least 2^{18} processors. The Paris interface includes software support for a *virtual processor facility*, which multiplexes the processors (dividing up the memory accordingly) to simulate a machine with a multiple of the number of physical processors. The ratio is called the *virtual processor ratio*, and most parallel operations will be slowed down by this ratio. However, because of the operation in the heavily-loaded limit, very few operations (some communication algorithms excepted) will be slowed down by more than the virtual processor ratio. Accordingly, many of our timing results are done with 8K processors, and timing results for a 64K processor machine are given by extrapolation.

Since a given problem may consist of many data sets which are successively mapped onto the Connection Machine processors during the computation, Paris supports the notion of the *virtual processor set*, or VP set. A VP set is a set of (virtual) processors onto which a given data set is mapped. There may be different VP sets for different data structures existing at the same time. The physical processors used to implement the VP sets will not be disjoint. At any given time, at most one particular VP set is selected and active; however, that VP set may communicate with other VP sets by sending data with destinations in a non-active VP set.

4 Parallel Algorithm

In this section we describe in more detail the implementation of one of the two data parallel algorithms for geometric hashing described in [9]. A number of alternative implementations are discussed at the end of the section. In the current system, the database contains M models (M is typically 1024) that are synthesized randomly: for each model m we generate a set $S_m = \{(x_k, y_k)\}_{k=1}^{n}$ of n coordinate pairs distributed either according to a uniform distribution over the unit disc or a Gaussian distribution over $\mathcal{R}^2$. The number n of points per model is typically 16.

4.1 Preprocessing Phase

During the preprocessing phase the hash table data structure is constructed. This phase is performed off-line. The hash table corresponding to a given database of model objects is built only once and then stored in the DataVault. This phase is computationally expensive; the serial complexity is $\Theta(M n^3)$ for two point bases. Our parallel implementation has complexity $\Theta(M \log n)$ with $M n^3$ processors. With fewer processors the time complexity is higher; however, this is not of concern since the operation is performed off-line and only once.

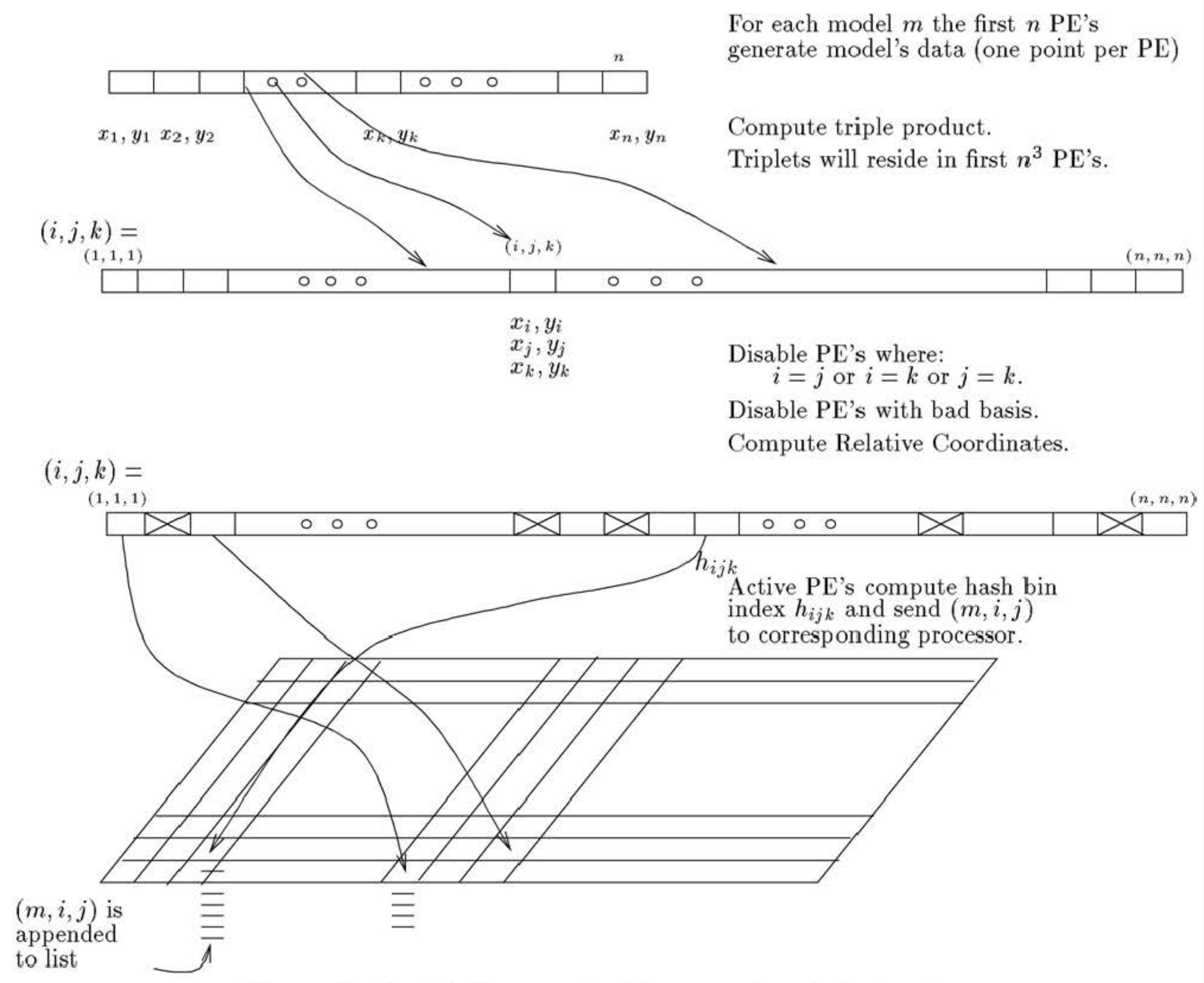

For each model m the first n PE's generate model's data (one point per PE)
n
x_1, y_1 x_2, y_2
x_k, y_k
x_n, y_n
Compute triple product. Triplets will reside in first n^3 PE's.
$(i, j, k) =$
$(1,1,1)$
(i,j,k)
(n,n,n)
x_i, y_i
x_j, y_j
x_k, y_k
Disable PE's where: $i = j$ or $i = k$ or $j = k$.
Disable PE's with bad basis.
Compute Relative Coordinates.
$(i, j, k) =$
$(1,1,1)$
(n,n,n)
h_{ijk}
Active PE's compute hash bin index h_{ijk} and send (m, i, j) to corresponding processor.
(m, i, j) is appended to list

Figure 5. Parallel Preprocessing Phase; creating the hash table.

Figure 5 shows a schematic diagram of this phase. We assign one processor to each bin in the two-dimensional hash table. In our current implementation, each processor associated with a hash table bin maintains local memory for the list of the $(model, i, j)$ tuples that correspond to the hash table entries. The purpose of the preprocessing phase is to construct these lists. Some of the resulting lists may be empty.

We iterate sequentially over the models. For each of the models, the lower order n processors generate the x and y coordinates of the n points in S_m; each processor generates the coordinates of exactly one such point. The set $(S_m \times S_m) \times S_m$ is then computed using the triple product algorithm of [9]. Each of the first n^3 processors now contains a triplet: $[(x_i, y_i), (x_j, y_j), (x_k, y_k)] \in (S_m \times S_m) \times S_m$. The first two points of each such triplet define a basis and thus a coordinate system, and we wish to compute the coordinates of the third point of the triplet with respect to that coordinate system. In order to avoid numerical overflows resulting from degenerate bases, we disable those processors where $i = j$, as well as those processors where the locations of the first two coordinate pairs are extremely close. In addition, to ensure that the third point is distinct from the first two, we also disable those processors where $i = k$ or $j = k$. The precise formulas for the computation of the relative coordinates depend on the type of the allowed transformations (e.g. similarity or rigid transformation). The active processors compute the coordinates of the third point using the appropriate function, and the index of the hash table bin (i.e., the address of the associated processor) where the tuple $(model, i, j)$ will be deposited. The rehashing transformations (Section 2) that allow us to achieve an even distribution of the $(model, i, j)$ tuples over the hash table, if desired, are also carried out during the hash bin index computation. A number of collisions are likely to occur at this stage when more than one PE attempt to deposit distinct tuples at the same hash table bin; we currently resolve the problem by using a simple message passing protocol that arbitrarily serializes the memory writes. Each tuple is appended to the list of the destination bin. There is some provision on some models of the Connection Machine for concurrent random access writes to different storage locations in each processor's local memory; however, such capabilities are limited, and we have not used them in this implementation.

This process is repeated for each of the models in the database. When all of the models have been processed the hash table data structure is stored in the DataVault.

4.2 Recognition Phase

Figure 6 shows a schematic diagram of the parallel implementation of the recognition phase. Three sets of virtual processors participate in this phase, namely V_1, the feature coordinate set, V_2, the hash table set, and V_3, the histogram bin set. We assume that a set of N (N is typically 200) interest points have been extracted from the scene and their coordinates reside in the local memory of the first N processors of virtual processor set V_1. The hash table is preloaded from the DataVault into the local memory of V_2.

In the first stage, the front end selects a basis pair (probe) and broadcasts the coordinates of the points in the basis to the N processors of V_1. Recall that each processor in V_1 already holds the coordinates of an interest point. The two processors whose interest points coincide with the broadcasted basis endpoints are disabled. The remaining processors of the virtual processor set V_1 compute the relative coordinates of the corresponding interest point with respect to the broadcasted basis. They then compute the index of the hash table bin to be notified. The rehashing mapping of Section 2 is utilized if the hash table creation incorporated rehashing. The operations are extremely fast since they involve minimal data movement.

In the second stage, messages saying "you receive one vote" are sent by the active processors of V_1 to the hash bins. Each hash bin is assigned to a virtual processor from the set V_2. The messages are sent using additive writes and general routing; multiple votes destined for the same recipient processor combine in the routers. Each of the processors of the set V_2 maintains a local "counter" variable in order to keep track of the number of votes the processor receives.

In the last stage, the processors of the set V_2 that received one or more messages scan their lists

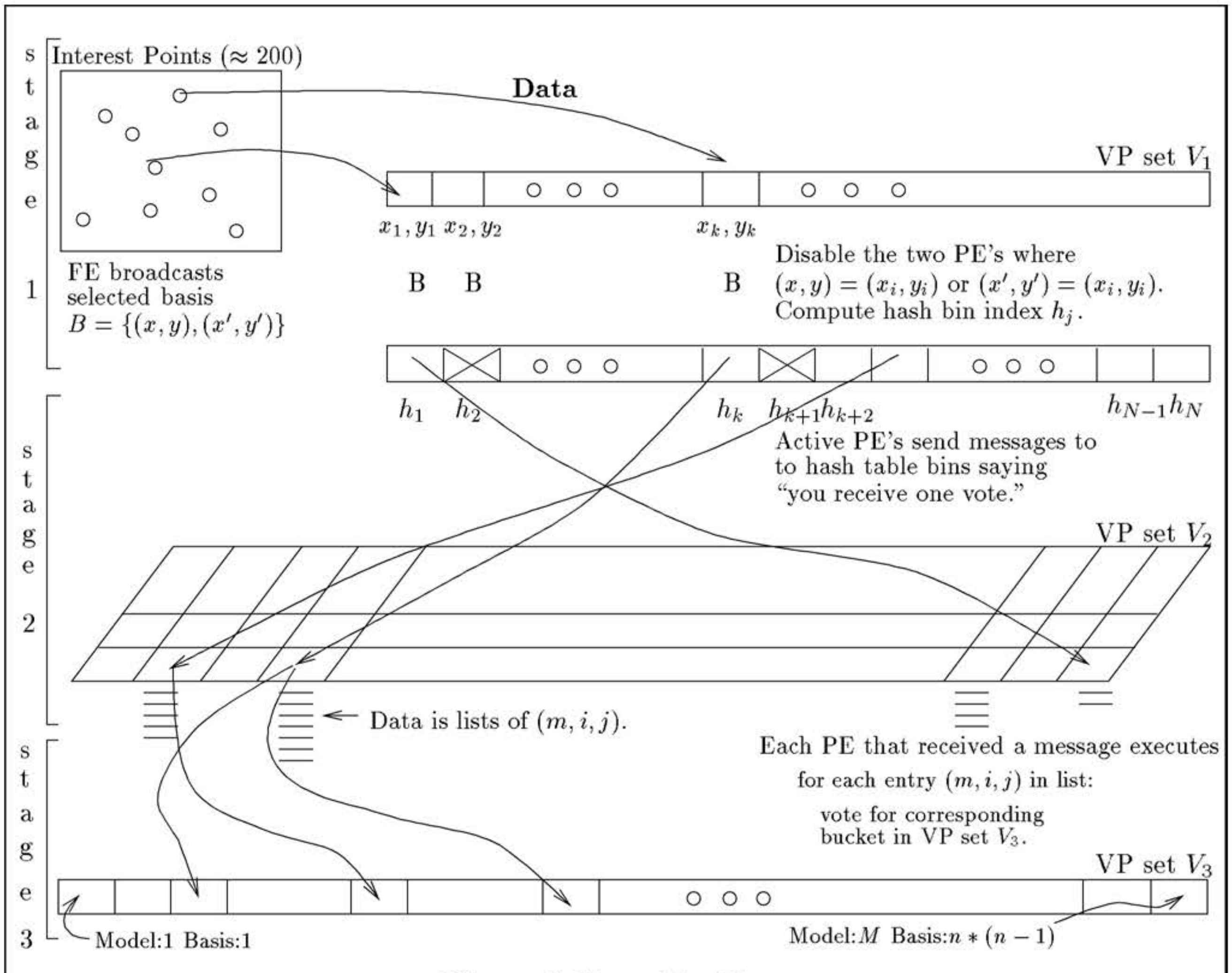

stage 1
Interest Points (≈ 200)
Data
VP set V_1
x_1, y_1 x_2, y_2
x_k, y_k
FE broadcasts selected basis $B = \{(x,y),(x',y')\}$
B B B
Disable the two PE's where $(x,y)=(x_i,y_i)$ or $(x',y')=(x_i,y_i)$. Compute hash bin index h_j.
h_1 h_2 h_k h_{k+1} h_{k+2} h_{N-1} h_N
stage 2
Active PE's send messages to to hash table bins saying "you receive one vote."
VP set V_2
Data is lists of (m,i,j).
stage 3
Each PE that received a message executes
for each entry (m,i,j) in list:
vote for corresponding bucket in VP set V_3.
VP set V_3
Model:1 Basis:1
Model:M Basis:$n*(n-1)$

Figure 6. Recognition Phase.

of tuples and vote for each of the $(model, i, j)$ entries by sending messages to the proper processor (histogram bin) of virtual processor set V_3. These messages contain the value of the counter variable at the processor where they originate from. The messages are again additive writes and votes destined for the same recipient combine in the routers. The time needed for the traversal of the list is dominated by the longest such list, so it is desirable that the hash table be equalized using the rehashing functions of Section 2). The voting is equivalent to histogramming data which consist of $(\log M + 2 * \log n)$-bit words, and can be very expensive even when performed in parallel. In our case the entries are 18-bit items, and the virtual processor ratio is 4 on a 64K machine and 32 on an 8K one. Routing operations are costly at such high ratios. However, as noted in the previous section, improved histogramming methods could be utilized. Last, a global-max operation of the vote tallies of each of the processors of the set V_3 recovers the winning (m, i, j) combination. The model is m and the broadcasted basis corresponds to the basis defined by the i-th and j-th points of that model; from this last correspondence we can recover the transform that the model has undergone, and verify the quality of the match. If the quality of the match is poor, either because an unstable basis was selected, or because the basis did not belong to the model, a new basis pair is selected randomly from the scene and the above operations are repeated.

Clearly, it is very unlikely that the first probe will consist of a pair of points both belonging to the model embedded in the scene. Thus, a number of probes will be required before a basis with both points on the model is employed. A number of heuristics facilitate the probe selection.

Let us assume that a model consisting of n points is embedded in a scene of S points. A basis pair (probe) is *admissible* iff its length is in the interval $[l, L]$, $l, L \in \mathcal{R}$. The admissibility criterion implicitly assumes that no embedded model will have a size that is smaller (respectively larger) than a certain portion of the scene; in typical scenes the portion occupied by the embedded model ranges between 1/16 and 1/4 of the scene area. Out of $S(S-1)$ possible bases, $n(n-1)$ can be formed by using points on the model. Use of the admissibility criterion reduces these figures to $\delta S(S-1)$ and $\epsilon n(n-1)$ where $0 < \delta, \epsilon \leq 1$. Assuming a uniform distribution for both the scene and the model points, the typical values for δ and ϵ are 0.3 and 0.8 respectively. Using standard probability theory it can be easily seen that the probability ρ of randomly selecting t consecutive basis pairs not belonging to the model is $\prod_{i=0}^{t}(1 - \epsilon n(n-1)/(\delta S(S-1)))$. For models of $n = 16$ points that are embedded in scenes of $S = 200$ points, the probability of $t = 250$ consecutive admissible probes that do not belong to the embedded model is $\rho \approx 1.5\%$. In our simulations, no more than half as many probes were ever required, i.e. considerably fewer than the 11940 possible ones.

4.3 Alternative Implementations

As we have already indicated, the last stage of the recognition phase constitutes a histogramming operation, and can be implemented by a number of methods [9]. Since this voting process currently accounts for 99% of the execution time of the recognition phase, efficiencies in histogramming would give much improvement to the performance of the implementation. Use of radix sort in conjunction with a sorting-based histogramming [9] would very likely reduce processing times, at least for our instances of 18-bit words, and typical sequences of 256K items to be histogrammed.

In the current implementation we use only one basis pair at a time during the recognition phase. However, by repeating data (the coordinates of the S interest points) we can employ multiple bases at once. Assuming scenes with an average of 200 interest points, as many as 256 bases can be used at the same time on the 64K Connection Machine. Each hash table bin maintains a separate vote counter for each of the 256 bases. Again the messages sent to the bins are additive writes. However, each message must now be "tagged" with the identifier of the basis from which the message starts.

When multiple bases are employed, it might be necessary, during the second stage of the recognition phase (when sending messages to hash bins), to serialize over the different basis sets. However, this was not investigated in our experiments. A further complication arises here from the requirement that the recipient processors must all correspond to the same basis selection. A possible solution can be a finer hash table virtual processor set, where each bin's entries are shared among 256 processors, one per possible basis set; the messages from V_1 are then sent to the proper processor of the proper bin.

Another implementation can result from a different mapping of the data structures to processors. In the current implementation, we map the scene's interest points onto the processors of the set V_1; the coordinates of the selected basis are broadcasted by the front end to those processors. Alternatively, each processor of V_1 could be made responsible for one basis pair. The coordinates of the *interest points* could either be broadcasted by the front end, or loaded from the DataVault. For scenes of approximately 200 points, assuming that each of the processors performs all the necessary computations for the entire scene based on the two-point basis, all possible bases can fit in a 64K Connection Machine at once.

We note that our implementation is at one extreme: data parallel over the points in the image, and serial over the basis sets, whereas the approach just described is at the other extreme: data parallel over the bases, but serial over the image points. In between, is the alternative mentioned earlier in this section, i.e. multiple bases at once.

5 Experiments

In this section we present the results of experiments that were carried out using databases of synthetic models. The model databases contained either 512 or 1024 models, with each of the models consisting of 16 randomly generated points. These random points are distributed either uniformly over the unit disc, or according to a Gaussian distribution of mean zero and standard deviation 1. The models are allowed to undergo either a similarity or a rigid transformation. All of the experiments were carried on an 8K processor Connection Machine system. Figure 7 shows some typical models (dot patterns). The

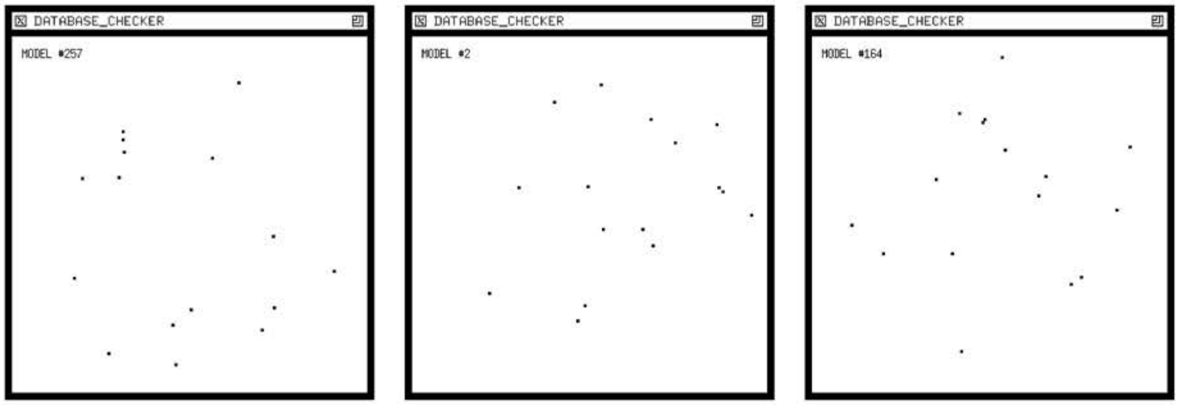


Figure 7. Typical models.

experiments are divided into two categories: (a) performance evaluation experiments and (b) statistical behavior evaluation experiments. The performance evaluation experiments attempted to determine the real time performance of the implementation we described in Section 4. The statistical behavior experiments aimed at examining (i) the filtering power of the method and (ii) its statistical behavior in the presence of noise in the input. In both cases all four combinations were studied:

(1) Rigid Transformations and Gaussian distributed model points;

(2) Similarity Transformations and Gaussian distributed model points;

(3) Rigid Transformations and uniform distribution of the model points over the unit disc; and,

(4) Similarity Transformations and uniform distribution of the model points over the unit disc.

5.1 Performance Evaluation

In this section, we describe our experience with the real time performance of the implementation as given in Section 4. The databases contained 1024 synthetically generated models. In addition to the four combinations above, the variants concerning re-quantization of hash space variants for combinations (1), (2), and (3) were also examined. The timings were performed using calls to the `CM-time()` routine of the PARIS interface.

The scenes for the experiments were generated synthetically. In order to create a scene, one of the models is selected from the database; after an arbitrary rotation and translation (and possibly scaling), the model is embedded in a scene of randomly generated points. The total number of points in the final scene is approximately 200; of these, 16 belong to the model. Noise is added to the positions of the points (through round-off error).

During the recognition phase, the front end randomly selects a pair of model points as the basis to be used. The computation proceeds as outlined in Section 4. We tested the implementation with a large number of scenes. For cases (1) through (4) above, the recognition process takes 3.4 seconds; between 7 and 10 milliseconds are spent on the computational part of the algorithm and the remainder of the time is devoted to the voting process. When we make use of the rehashing functions and of the appropriate uniform hash table, the recognition process is completed in 1.2 seconds with 8K processors. It must be stressed here that none of the above experiments makes use of the existing hash table symmetries; exploitation of those symmetries would speed up the recognition by a factor of 2, bringing the time required to perform recognition down to 0.60 seconds on an 8K Connection Machine. This is because only half of the hash table must be stored in the Connection Machine, and thus the length of the lists of entries will be halved. Use of the folding (thereby using basis sets instead of basis pairs) would incur even greater computational savings.

On a 64K machine, we get not only an 8-fold speedup due to the additional processors, but an additional speedup due to reduced contention in the voting process. (I.e., the routing algorithm on the smaller machine is less efficient, so that additional speedup is achieved because, in essence, the algorithm changes). The additional processors give us a full 8-fold speedup because the length of the lists of entries in the hash tables will be decreased proportionally. Extrapolations indicate that a 64K machine, using the symmetries and rehashings, but without taking advantage of the foldings, would process the entire computation, including the voting process, in at most 70 milliseconds.

Additional speedup can be obtained by making use of the folding of the hash table, thereby voting for basis sets as opposed to basis pairs, or by doing partial histogramming. For example, we experimented with the following modification. The model numbers require 10 bits of information; grouping models into 32 classes, a class of models may be specified with only 5 high-order bits of the model number. Instead of voting for the $(model, basis)$ pair, we vote for the $(model{-}class, basis)$, which requires 5 fewer bits. Although this potentially confounds 32 models into a single model, in our experiments, very few items received additional additional votes, and thus the resulting vote tallies were changed very little. From the $(model - class, basis)$ receiving the most votes, it is very easy to determine the corresponding best model. This modification resulted in execution times of 0.8 seconds for the recognition phase on an 8K machine (compared to the previous 1.7 second result). On a larger machine, improvements might well be similar.

Number of PE's	Symmetries	HT Equalization	Partial Histogramming	Real Time (per probe)
8 K	NO	NO	NO	3.4 sec
8 K	NO	YES	NO	1.2 sec
8 K	YES	NO	NO	1.7 sec $(*)$
8 K	YES	YES	NO	0.6 sec $(*)$
8 K	NO	NO	YES	0.8 sec
64 K	YES	YES	NO	$\leq$ 70 msec $(*)$

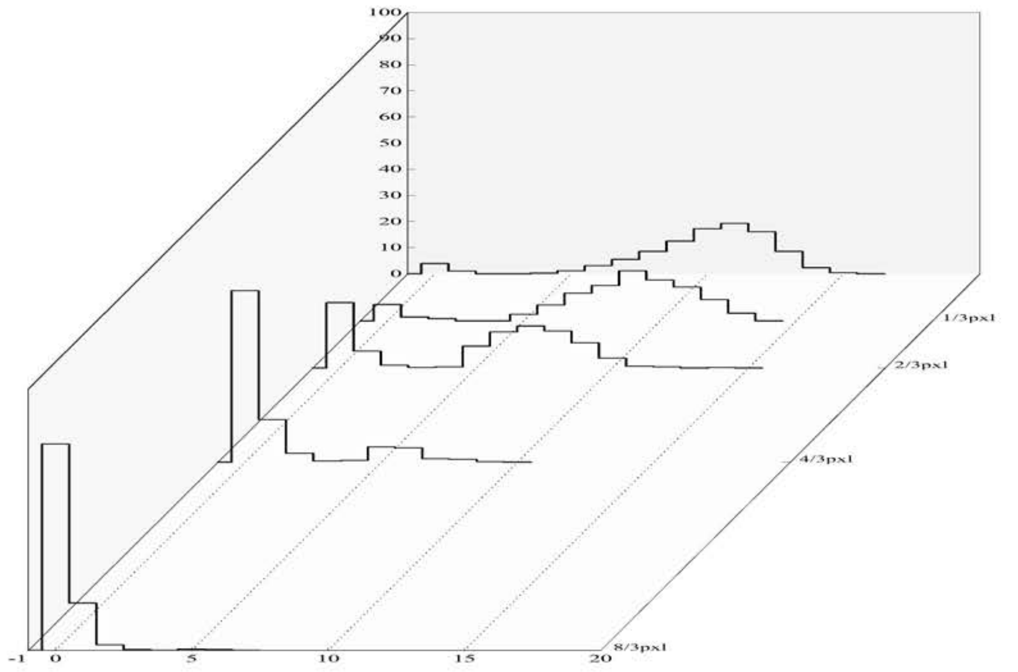
Table 1. Table of timing results. Values are given for implementations with and without use of foldings and/or symmetries. Items marked with a $(*)$ indicate that the value has been obtained by means of extrapolation.

Such model-binning is equivalent to performing an iterated histogramming of the model and basis pairs. At an opposite extreme, the radix sorting method of histogramming should prove to be more efficient than the implementations described here for the range of sizes of the database used in our experiments. Table 1 summarizes the above results.

5.2 Method Statistics

In this section, we report on a different type of experiments that examined (i) the performance of the geometric hashing technique in the presence of positional error in the scene features, and (ii) its power as a filtering method. The databases contained 512 models. Due to restrictions in the size of the hash table, databases corresponding to the models whose points were Gaussianly distributed contained 96% of all possible entries. On the other hand, the databases corresponding to the models whose points were distributed uniformly over the unit disc contained 100% of the possible hash table entries. In all our experiments, and for each computed pair of relative coordinates, we accessed a single hash table bin during the recognition phase (instead of a range of bins as suggested in [5]).

For basis pairs belonging to the model embedded in the scene, we examined the effect that noise in the input has on the number of votes received by the probed basis. For each of the four cases, the experiments were carried out for a number of different scenes and for a number of different models. In particular, 10 models were extracted from each of the four databases and were subsequently embedded in a scene containing uniformly distributed random points. The models were embedded so that they occupied between 1/16 and 1/9 of the scene area, which was 350×350 pixels. Each scene consisted of 200 points, 16 of which belong to the embedded model. The positions of the feature points in each of the 40 scenes were then perturbed by Gaussian noise. The value of σ, the standard deviation of the



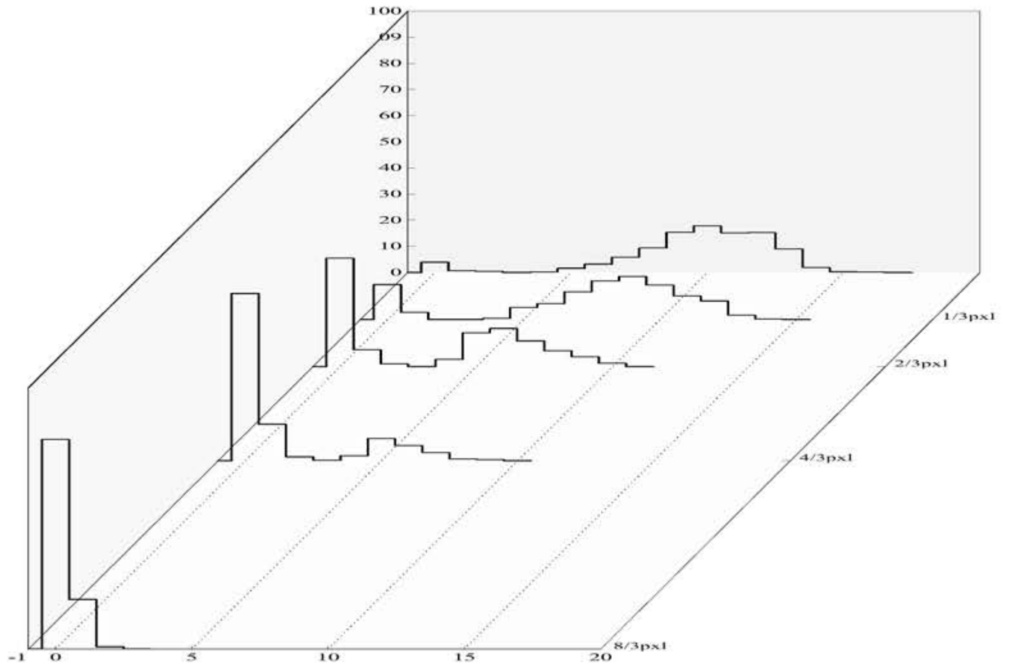
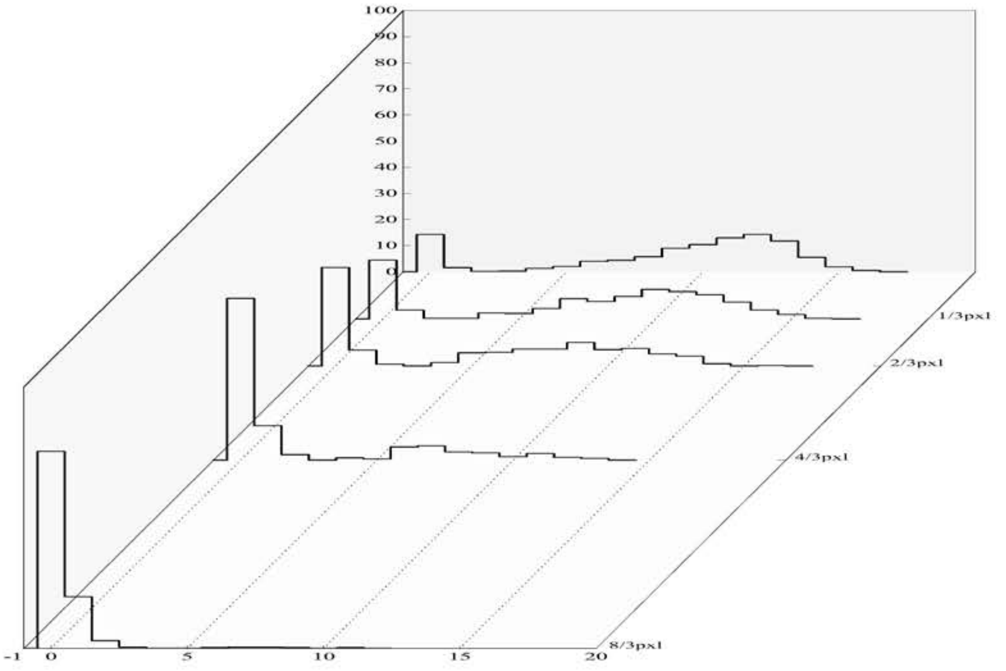


Figure 8. Percentage of model bases that receive k votes when used as probes, for different amounts of Gaussian noise. The models can only undergo rigid transformations. Top: the models points are distributed according to a Gaussian of $\sigma = 1$. Bottom: the models points are distributed uniformly over the unit disc.



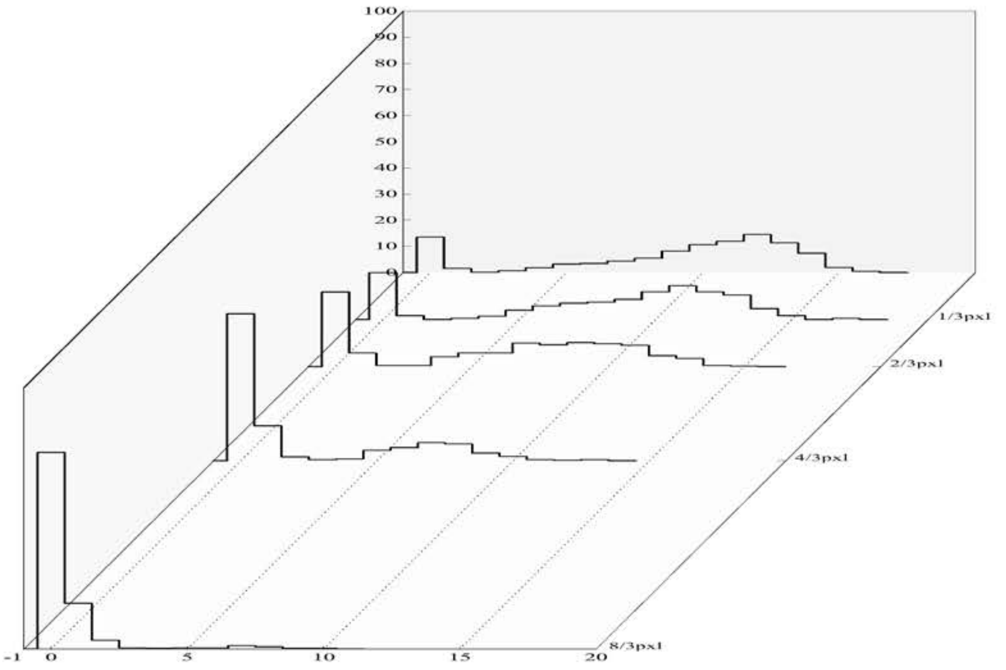


Figure 9. Percentage of model bases that receive k votes when used as probes, for different amounts of Gaussian noise. The models can only undergo similarity transformations. Top: the models points are distributed according to a Gaussian of $\sigma = 1$. Bottom: the models points are distributed uniformly over the unit disc.

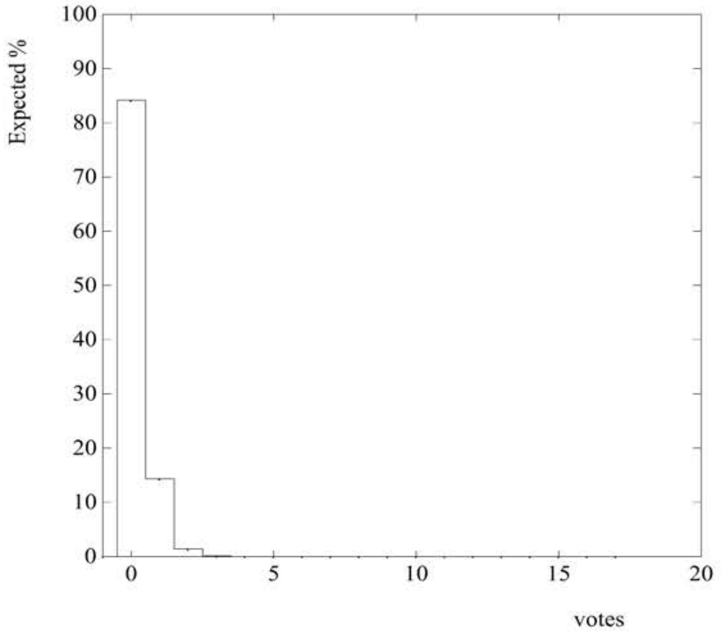
Gaussian noise, was such that $\sigma = 0, 1/3, 2/3, 4/3,$ or $8/3$ pixels. Every one of the 120 distinct pairs was used as a basis. For each of the five values of σ, the obtained results were averaged over the 10 different scenes. For the combinations $(1)-(4)$, Figures 8 and 9 show the percentage of the probed bases which receive k votes $k = 1, \ldots 14$, for the different values of σ.

As can be seen from these Figures, values of $\sigma \leq 1/3$ pixels results in a peak around roughly 11 votes. Noise at the high end ($\sigma \geq 4/3$ pixels suffices to make the object practically undetectable. However, this is expected since the maximum acceptable amount of noise that will permit detection of the hidden model is directly related to the resolution of the hash table, i.e. the size of the hash table bins. The larger the size of the bins the larger the tolerance to noise in the input; however, large size bins (i.e. coarse quantization) result in insufficient discriminatory power among the models, for certain choices of features in the scene. In this case, the hash table will yield too many candidate matches (under certain circumstances), and one must revert to a situation of testing and verifying many hypothesized matches. On the other hand, the geometric hashing search is highly parallel.

To allow for greater error tolerance, one needs to access a region of the hash table (range of hash table bins) instead of a single bin. However, the situation is complicated by the fact that the size of the region that needs to be accessed depends on the basis pair that is actually used as well as the values of the computed invariants. In the case of affine invariant point matching both the shape and orientation of the region (in addition to its size) depend on the basis selection.

Another question related to the performance of geometric hashing is the percentage of the model/basis combinations receiving k, $0 \leq k \leq n-2$, votes for a randomly selected basis pair (probe). The answer to this question is directly related to the filtering power of geometric hashing. Clearly, the smaller the percentage of model/basis combinations receiving above a certain number of votes, the higher the filtering power of the method and the smaller the cost of the verification step [4, 6]. Again 10 models were selected for each one of the combinations (1)-(4), and embedded in a scene of uniformly distributed random points. The embedded models occupied roughly between 1/16 and 1/9 of the scene's area. The number of feature points in the final scene was 200. For each of the test scenes, we randomly selected 480 probes as follows: 240 basis pairs were formed using one model and one clutter point, and 240 basis pairs were formed using uniquely clutter points. For each probe the number of model/basis combinations that received between 0 and 16 votes was recorded and the results for each scene were combined. Figures 10 through 11 graphically show the average over the 10 test scenes for each one of the combinations (1)-(4).

We see that fewer than a fraction of 1% of all possible combinations receive more than 5 votes in the case of rigid transformations. For the case of similarity no more than 4.5% of all possible such combinations receive more than 7 votes. On the basis of these figures, we see that for the similarity case, noise in the region of $\sigma = 1/3$ pixels is tolerable for filtering out roughly 95% of all possible model/basis combinations. Since $\sigma = 1/3$ corresponds to a ± 1 pixel variation (at the maximum), we see that for models of size roughly 80×80 pixels, with basis lengths of roughly 40-50 pixels, the method yields reasonable filtering power in the presence of ± 1 pixel positional error.



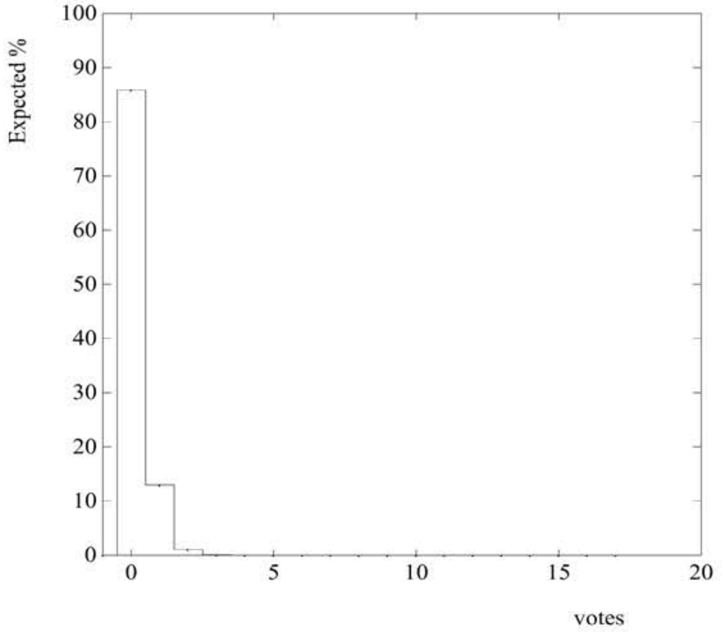
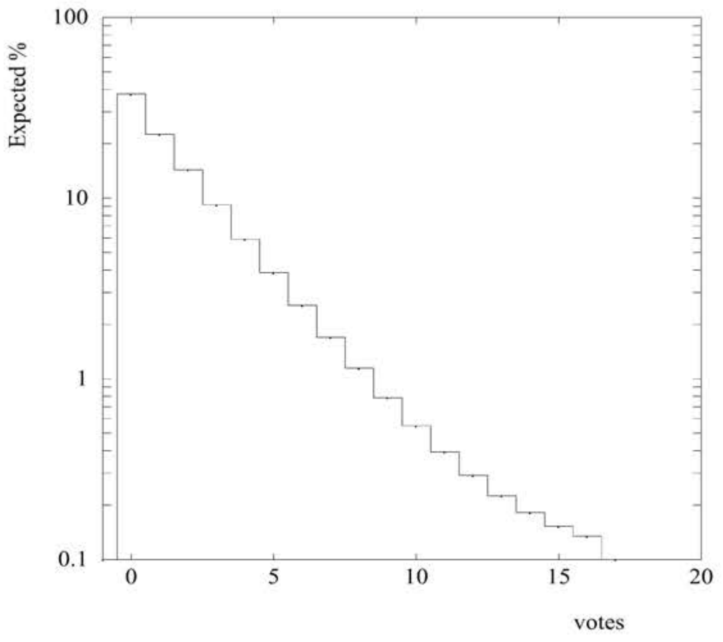


Figure 10. Rigid Transformations: expected percentage of model/basis combinations receiving exactly k votes. Top: the model points were distributed according to a Gaussian of $\sigma = 1$. Bottom: the models points are distributed uniformly over the unit disc.



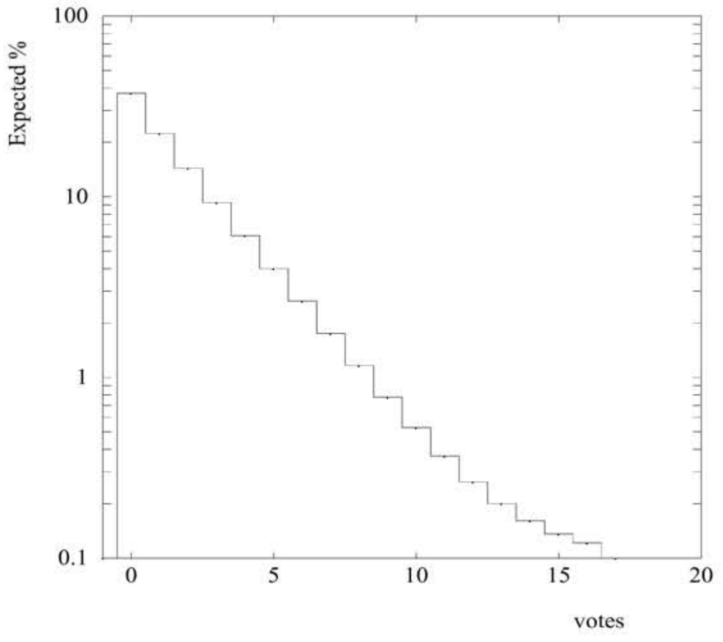


Figure 11. Similarity Transforms: expected percentage of model/basis combinations receiving exactly k votes. Top: the model points are distributed according to a Gaussian of $\sigma = 1$. Bottom: the models points are distributed uniformly over the unit disc.

References

[1] **Chin, R. and C. Dyer**. "Model-Based Recognition in Robot Vision". *ACM Computing Surveys*, 18(1):67–108, March 1986.

[2] **Hummel, R. and H. Wolfson**. "Affine Invariant Matching". In *Proceedings of the DARPA Image Understanding Workshop*, April 1988.

[3] **Huttenlocher, D. P. and S. Ullman**. "Object Recognition by Affine Invariant Matching". In *Proceedings of the DARPA Image Understanding Workshop*, April 1988.

[4] **Lamdan, Y. and H. Wolfson**. "Geometric hashing: A General and Efficient Model-Based Recognition Scheme". In *Proceedings of the 2nd International Conference on Computer Vision*, pages 238–249, 1988.

[5] **Lamdan, Y. and H. Wolfson**. "On the Error Analysis of Geometric Hashing". Technical Report 213, Robotics Lab., New York University, October 1989.

[6] **Lamdan, Y. and J. Schwartz and H. Wolfson**. "Object Recognition by Affine Invariant Matching". In *Proceedings of the IEEE Computer Vision and Pattern Recognition Conference*, pages 335–344, Ann Arbor, Michigan, June 1988.

[7] **Lowe, D. G.** *Perceptual Organization and Visual Recognition*. Kluwer Academic Publishers, 1985.

[8] **Press, W. H., B. Flannery, S. A. Teukolsky and W. T. Vetterling**. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1988.

[9] **Rigoutsos, I. and R. Hummel**. "Scalable Parallel Geometric Hashing for Hypercube SIMD Architectures". Technical Report 553, Courant Institute of Mathematical Sciences, New York University, April 1991. Submitted for publication to *IEEE Computer: Special Issue on Parallel Algorithms for Computer Vision and Image Understanding*.